

INFORMATICA

Clase de hoy

- Programación modular. Subprogramas:
 - Funciones
 - Subrutinas

Programación modular

- Motivación: Ej. Ordenar filas de matriz.

```
program principal
```

```
...
```

```
do i=1,n_filas
```

```
    V = A(i,:)
```

```
    ! ordeno el vector V
```

```
    A(i,:) = V
```

```
end do
```

Programación modular

- Técnica de programación que consiste en dividir el código en *unidades de programa*

Tipos de unidades de programa.

- Programa principal: `program`
- Subprogramas:
 - Función: `function`
 - Subrutina: `subroutine`

Programación modular

Una unidad de programa es un conjunto independiente de instrucciones lógicas al que se le asigna un nombre para identificarlo.

- Solo puede haber un programa principal, que controla y relaciona a los demás.
- Cada unidad de programa debe presentar una estructura lógica coherente y resolver una parte bien definida del problema.
(Finalidad e independencia)

- Hallar la inversa de una matriz
- Calcular la distancia entre dos puntos
- Ordenar un vector

Programación modular

VENTAJAS

- Los programas son más sencillos de escribir y de depurar. Se puede probar cada unidad por separado.
- Los programas son fácilmente ampliables y la programación colaborativa. (cada programador hace una parte).
- Un mismo subprograma se puede reutilizar, llamándolo varias veces dentro del mismo programa principal ó agrupándolos en **librerías** que se usarán en diferentes programas.

Programación modular

Ejemplo: Subprograma subrutina que ordena un vector de enteros

```
subroutine ordenar(Vector, n)

    integer,intent(in)           :: n
    integer,intent(inout)        :: Vector(n)

    integer :: i, j, temp

    do i = 1,n - 1
        do j = i, n

            if (Vector(j) > Vector(i)) then
                temp = Vector(i)
                Vector(i) = Vector(j)
                Vector(j) = temp
            end if

        enddo
    enddo

end subroutine ordenar
```

Programación modular

Ejemplo: Subprograma subrutina que ordena un vector de enteros



```
subroutine ordenar(Vector, n)
```

```
integer,intent(in)          :: n
```

```
integer,intent(inout)       :: Vector(n)
```

```
integer :: i, j, temp
```

```
do i = 1,n - 1
```

```
do j = i, n
```

```
if (Vector(j) > Vector(i)) then
```

```
temp = Vector(i)
```

```
Vector(i) = Vector(j)
```

```
Vector(j) = temp
```

```
end if
```

```
enddo
```

```
enddo
```




```
end subroutine ordenar
```



Programación modular

Ejemplo: Subprograma subrutina que ordena un vector de enteros

subroutine ordenar(Vector, n)  argumentos

Declaración de  argumentos

```
integer, intent(in)          :: n  
integer, intent(inout)       :: Vector(n)  
  
integer :: i, j, temp
```

```
do i = 1, n - 1  
  do j = i, n  
  
    if (Vector(j) > Vector(i)) then  
      temp = Vector(i)  
      Vector(i) = Vector(j)  
      Vector(j) = temp  
    end if
```

```
  enddo  
enddo
```

```
end subroutine ordenar
```

Programación modular

Ejemplo: Subprograma subrutina que ordena un vector de enteros

```
subroutine ordenar(Vector, n)  
  
integer, intent(in)           :: n  
integer, intent(inout)       :: Vector(n)
```

in Atributos de los argumentos:

do `intent(in)`: El argumento es de entrada y su valor no se puede modificar dentro de la subrutina.

`intent(inout)`: El valor puede cambiar durante la ejecución de la subrutina

`intent(out)`: El argumento no se puede usar hasta que no se le asigna valor dentro de la subrutina.

en

```
end subroutine ordenar
```

Programación modular

Ejemplo: Subprograma subrutina que ordena un vector de enteros

```
subroutine ordenar(Vector, n)

    integer,intent(in)          :: n
    integer,intent(inout)       :: Vector(n)

    ← integer :: i, j, temp

    do i = 1,n - 1
        do j = i, n

            if (Vector(j) > Vector(i)) then
                temp = Vector(i)
                Vector(i) = Vector(j)
                Vector(j) = temp
            end if

        enddo
    enddo

end subroutine ordenar
```

Variables
locales

Programación modular

Ejemplo: Subprograma subrutina que ordena un vector de enteros

```
subroutine ordenar(Vector, n)

    integer, intent(in)          :: n
    integer, intent(inout)       :: Vector(n)

    integer :: i, j, temp

    do i = 1, n - 1
        do j = i, n

            if (Vector(j) > Vector(i)) then
                temp = Vector(i)
                Vector(i) = Vector(j)
                Vector(j) = temp
            end if

        enddo
    enddo

end subroutine ordenar
```

Cuerpo de la
subrutina

Programación modular

Ejemplo: Llamada desde el principal

```
program principal

    integer      :: A(5,3)
    integer      :: i, j

    do i = 1,5
        do j = 1,3
            A(i,j) = i**2 +j
        enddo
    enddo

    ! Ordenar las filas de la matriz

    do i = 1,5
        call ordenar(A(i,:),3)
    enddo

end program
```

Programación modular

Ejemplo: Llamada desde el principal

```
program principal

    integer      :: A(5,3)
    integer      :: i, j

    do i = 1,5
        do j = 1,3
            A(i,j) = i**2 +j
        enddo
    enddo

    ! Ordenar las filas

    do i = 1,5
        call ordenar(A(i,:),3)
    enddo

end program
```

```
subroutine ordenar(Vector, n)

    integer,intent(in)      :: n
    integer,intent(inout)   :: Vector(n)

    integer :: i, j, temp

    do i = 1,n - 1
        do j=i,n
            ...
        enddo
    enddo

end subroutine ordenar
```

Programación modular

Ejemplo: Llamada desde el principal

```
program principal

  integer      :: A(5,3)
  integer      :: i, j

  do i = 1,5
    do j = 1,3
      A(i,j) = i**2 +j
    enddo
  enddo

  ! Ordenar las filas

  do i = 1,5
    → call ordenar(A(i,:),3)
  enddo

end program
```

```
subroutine ordenar(Vector, n)

  integer,intent(in)      :: n
  integer,intent(inout)   :: Vector(n)

  integer :: i, j, temp

  do i = 1,n - 1
    do j=i,n
      ...
    enddo
  enddo

end subroutine ordenar
```

Programación modular

Ejemplo: Llamada desde el principal

```
program principal
```

```
integer      :: A(5,3)
```

```
integer      :: i, j
```

```
do i = 1,5
```

```
do j = 1,3
```

```
    A(i,j) = i**2 +j
```

```
enddo
```

```
enddo
```

```
! Ordenar las filas
```

```
do i = 1,5
```

```
→ call ordenar(A(i,:),3)
```

```
enddo
```

```
end program
```

```
subroutine ordenar(Vector, n)
```

```
integer,intent(in)      :: n
```

```
integer,intent(inout)   :: Vector(n)
```

```
integer :: i, j, temp
```

```
do i = 1,n - 1
```

```
do j=i,n
```

```
...
```

```
enddo
```

```
enddo
```

```
end subroutine ordenar
```

COPIA

Programación modular

Ejemplo: Llamada desde el principal

```
program principal

  integer      :: A(5,3)
  integer      :: i, j

  do i = 1,5
    do j = 1,3
      A(i,j) = i**2 +j
    enddo
  enddo

  ! Ordenar las filas

  do i = 1,5
    call ordenar(A(i,:),3)
  enddo

end program
```

```
subroutine ordenar(Vector, n)

  integer,intent(in)      :: n
  integer,intent(inout)   :: Vector(n)

  integer :: i, j, temp

  do i = 1,n - 1 ←
    do j=i,n
      ...
    enddo
  enddo

end subroutine ordenar
```

Programación modular

Argumentos: El paso de *argumentos por cabecera* intercambia información entre los distintos subprogramas del código.

La regla general que rige la asociación entre los argumentos verdaderos (programa principal) y argumentos ficticios (*dummy*) (en el subprograma) impone que ambos deben coincidir en:

- Tipo y *kind*
- Número
- Orden
- ~~Nombre~~

```
subroutine ordenar(Vector, n)
                        ↑      ↑
                        ↓      ↓
call ordenar(A(i,:), 3)
```

Programación modular

Argumentos: Si el argumento pasado es de tipo array no es necesario (si recomendable) indicar su tamaño.

```
subroutine ordenar(Vector)

    integer,intent(inout)      :: Vector(:)

    integer :: n
    integer :: i, j, temp

    n = size(Vector,1)
    do i = 1,n - 1
        do j = i, n

            ...

        enddo
    enddo

end subroutine ordenar
```

Programación modular

Ejemplo: Implementación (¿Dónde ponemos las subrutinas?)

```
program principal

    integer      :: A(5,3)
    integer      :: i, j
    ...
    ! Ordenar las filas
    do i = 1,5
        call ordenar(A(i,:))
    enddo

end program
contains
    subroutine ordenar(Vector)

        integer,intent(inout) :: Vector(:)

        ...

    end subroutine ordenar
end program
```

De momento (**HASTA QUE VEAMOS MÓDULOS**)

las escribiremos a continuación del programa principal.



Programación modular

Tipos de unidades de programa.

- Programa principal: `program`
- Subprogramas:
 - Función: `function`
 - Subrutina: `subroutine`

Programación modular

Tipos de unidades de programa.

- Programa principal: `program`
- Subprogramas:
 - Función: `function`
 - Subrutina: `subroutine`

Programación modular

- Subprogramas: `function`

Una función es un subprograma que:

- Indicándole desde la unidad de programa llamadora los datos de entrada (*argumentos*) con los que se desea que realice los cálculos,
- Devuelve una *única* variable a la unidad de programa llamadora.

```
function valor_abs(x)
  real, intent(in)  :: x
  real              :: valor_abs

  valor_abs = x
  if (valor_abs < 0.0) valor_abs = -valor_abs

end function valor_abs
```

Programación modular

```
program valor

    real          :: a,b,c

    a = -3.0
    b = valor_abs(a)

    c = 1.7
    write(*,*) valor_abs(c)

contains

    function valor_abs(x)
        real, intent(in) :: x
        real              :: valor_abs

        valor_abs = x
        if (valor_abs < 0.0) valor_abs = -valor_abs

    end function valor_abs

end program
```


Programación modular: Ejemplo

Ejemplo

Caso 1

- Escribir un subprograma **subroutine** que, dado un vector x , calcule su media y desviación típica.

Caso 2

- Escribir un subprograma **function** que, dado un vector x , calcule su media.
- Escribir un subprograma **function** que, dado un vector x y su media, calcule su desviación típica.

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i, \quad dt = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$$

Programación modular. Trabajo para casa

Escribir una subrutina que calcule el producto de dos matrices cualesquiera. La subrutina debe comprobar que las dimensiones de las matrices permiten multiplicarlas. La cabecera de la subrutina será:

```
subroutine producto (A, B, AxB, info )  
  real , intent (in) :: A(: , :)  
  real , intent (in) :: B(: , :)  
  real , intent (out) :: AxB (: , :)  
  integer, intent(out):: info  
end subroutine producto
```

Info contiene información sobre el resultado de la multiplicación: se ha podido llevar a cabo, las dimensiones de A y B no son las adecuadas... El significado de los distintos valores deberá estar comentado en el código.

Escribe otra subrutina que, mediante llamadas a la anterior, halle la potencia p-ésima de una matriz cuadrada.